DOCUMENT RESUME

ED 060 622 EM 009 628

AUTHOR Bork, Alfred M.
TITLE Conversion of PCDP Dialogs.
INSTITUTION California Univ., Irvine. Physics Computer Development Project.
SPONS AGENCY National Science Foundation, Washington, D.C.
PUB DATE 2 Dec 71
NOTE 7p.

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS *Computer Assisted Instruction; *Computer Programs; Physics Instruction; Program Descriptions; Programed Materials; Programing; *Programing Languages; Programing Problems
IDENTIFIERS PCDP; Physics Computer Development Project

ABSTRACT

An introduction to the problems involved in conversion of computer dialogues from one computer language to another is presented. Conversion of individual dialogues by complete rewriting is straightforward, if tedious. To make a general conversion of a large group of heterogeneous dialogue material from one language to another at one step is more ambitious. Three possible approaches are seen. Original programs might be fed to some kind of interpretive processor. Or source programs might be read by a background program in some language, then converted to binaries and load modules for the new language. Finally, an entire editing program could be written to convert autonomously, but this task might in the end be too difficult or too constricting to further change. (RB)

Conversion of PCDP Dialogs

Alfred Bork
University of California, Irvine

December 2, 1971

During the past few months we have talked with many people about the possibility of converting the dialogs developed by the Physics Computer Development Project to other machines. Our dialogs were written for the XDS Sigma 7, operating under BTM and UTS, so will not run directly on any other computer.

Since we find ourselves saying the same things to different people, we thought it would be best to put some of this material in writing, to serve as an introduction to the problems involved in attempting the conversion to a different facility. The teaching programs themselves, and our software, are described in the PCDP progress report and other literature, available upon request.

Individual Dialog Conversion

A number of our dialogs have been converted to other systems on an individual basis, by simply working from our existing flowcharts and/or programs, in rewriting the material in some other appropriate language. The dialog that has been most heavily worked this way is the conservation of energy dialog, CONSERVE, which now exists in about six versions.

Not very much in general can be said about such single-dialog conversion, because the process depends on the language in which the new program is to be written. That language must certainly have powerful and efficient string matching facilities, the ability to pick a string out of a larger string. It should also be capable of altering strings--removing blanks, replacing characters, etc. The flowcharts that are available for some dialogs, and that hopefully will be available for others later, give some clue as to how

d with many people about the
veloped by the Physics
chines. Our dialogs were
nder BTM and UTS, so will

things to different people,
of this material in writing,
ems involved in attempting
The teaching programs
bed in the PCDP progress
pon request.

ted to other systems on an
our existing flowcharts
al in some other appropriate
heavily worked this way
NSERVE, which now exists

out such single-dialog con-
the language in which the
guage must certainly have
facilities, the ability to
It should also be capable
eplacing characters, etc.
ome dialogs, and that hope-
r, give some clue as to how

to go about doing this. Our own programs are also very useful for such conversion efforts because the macros we use do not include any abbreviations; they are readable with only a minimum amount of practice on the part of others not initially familiar with our procedure.

It should be noted that the main difficulty in such conversions is likely to come with formula matching. Here the techniques which can be used tend to be language dependent. A program that depends heavily on being able to recognize the bewildering variety in which a formula comes in, as with many of our dialogs, succeeds or fails depending on how sophisticated the program is in this regard. It should be noted that formulae in our dialogs, as well as in physics generally, include more than algebraic expressions. Provision must be made for dealing with derivatives, multi-variable names, subscripted quantities, etc. Formula matching techniques which consider only algebraic entities, such as numerical substitution, are likely to be inadequate in many places, although they will work for certain dialogs.

General Conversion

Conversion of individual dialogs is straightforward, although tedious. Many of the people we have talked to, however, are interested in a more ambitious attempt to convert a large group of our dialog material at one blow, perhaps even most of it. So most of the present discussion will be oriented toward such full or almost full conversion.

Although this material is contained elsewhere in our literature, we begin by reviewing the structure of our own programs as they run on the XDS Sigma 7. The source programs are collections of macro calls (Procedures in METASYMBOL, the XDS assembly language), using over 100 macros that we have developed for the purposes of computer based instruction. One possible macro is a call to a FORTRAN subroutine, so pieces of the final running program may have originated in FORTRAN, particularly if calculational needs of some complexity are involved.

Occasionally a program may also have a few direct assembly language instructions. But this is in general rare and usually represents a transitional stage before a new macro has been written to take care of whatever task is being covered.

A final program will be composed of a large number of source programs of (primarily) macro calls, perhaps as many as ten or fifteen. Each of these goes through the macro assembler (METASYMBOL) and leads to a binary. Then these binaries are put together by the loader to form a load module. Most of the programs are far too large to fit into the user area of core (many are more than 100K in length), so the load modules are usually elaborate overlay structures; some of the macros are designed to support overlay facilities. Thus the program the student calls is a load module. He is not aware that pieces are called in from the disk as he needs them.

Perhaps I should stress the reason for the macro approach, since that is not always clear to those unfamiliar with PCDP. We are not primarily interested in producing software. Every piece of software that we have developed has been in response to some teaching need. We never abstractly decide what facilities we want, but we develop teaching materials and then increase the facilities when new needs show up in such development. The macro procedure was adapted as being the one in which we could be most responsive to such pedagogical needs. We can easily add macros and expand the capability of older ones, so the software can respond to teaching demands.

Three Possible Approaches

At least three possibilities appear for large-scale conversion of the dialog material. First, it might be possible that our source programs would serve, perhaps with slight modifications, as input to an interpretive processor. Second, our source programs might be read by a background (or on-line) program in some language, say FORTRAN, PL/1, or BASIC and then converted into binaries and load modules for the particular system at hand. Third, it might be

ect assembly language
usually represents
en written to take

mber of source programs
ten or fifteen. Each
ASYMBOL) and leads
ther by the loader
re far too large to
than 100K in length),
lay structures; some
facilities. Thus
e. He is not aware
needs them.

cro approach, since
with PCDP. We are
e. Every piece of
esponse to some <u>teaching</u>
ties we want, but we
the facilities when
acro procedure was
most responsive to
cros and expand the
respond to teaching

e-scale conversion of
sible that our source
difications, as input
ource programs might
in some language, say
nto binaries and load
Third, it might be

possible to implement the entire macro structure in a macro facility of another computer.

This third possibility would probably also involve the construction of an editing program to accept our source programs, and modify them slightly, since every macro assembler has different conventions as to acceptable form. It would be possible to do such syntactical conversion by hand, but it would be more elegant and more practical to have the computer do this itself. The details of this editor would be dependent on the particular machine used, and how its macro assembly language differed from that of the Sigma 7.

In comparing these three possibilities it is clear that the second and the third would produce more efficient running code, since in each case the program the students use would be a load module and so would not have the overhead of an interpretive procedure.

Because of the differences of monitors, it is likely that some compromises will have to be made in the conversion process. Although most computers are similar in their architectural details, with the exception of a few machines, they differ in the range of services offered by the monitor. We have tried to use in our case everything that was useful in <u>our</u> teaching situations, and this has sometimes led us to do things which might not be possible to do in other systems. Likewise other systems might have features that lead to possibilities that we could not consider.

It would probably be worthwhile in the conversion process to have people intimately acquainted with both assembly languages and time-sharing monitors, in order to resolve questions of this kind.

We feel that at this stage of the game an interpretive system, or compiler for our own language, is perhaps unwarranted and too straightjacketed a situation. We want to maintain maximum flexibility. We also want to be able to do anything that is <u>doable</u> within the system, so that we do not preclude any particular ways computers can be used in the teaching environment. It may be obvious to you from the literature you have already seen from the project, but I thought it was worth stressing here. As our system evolves with usage, yours will too, hopefully.

Furthermore an interpretive approach is wasteful of computer time, by at least a factor of four, if the program is to be used with large numbers of students. So we do not recommend that approach, although it may be desirable in some situations.

Files

Improvement of computer-based educational materials is heavily dependent on selectively saving information on files for later examination by the author of the program. Experience in our project indicates that dialogs, when they are initially written, are almost always poor. It is only after a long period of use, and much student feedback, that we can improve them so that they function in the way we would like them to.

In our system the choice of what is saved is up to the author, with the use of SAVE or SAVEID commands. These can occur anywhere within the program. In each case we save identification telling where it is within the program (specified by the author), the time and date, the student's identification (if SAVEID is used), and the last input, including whatever processing on that input has taken place since it came in. The method of storage of this material should take into account that it will later be necessary to sort it on any of the interesting variables, and to print out various sorted lists.

Since this material is essential for the development of the dialog, great care must be taken so that as little as possible is lost. In our case if we attempt to write on a file currently open to another user, we wait a period of time and then proceed (for a finite number of times) to attempt to write again. Within the program we must examine the error code returned when a file error occurs; if

It may be
seen frcm the
As our system

computer time,
be used with large
approach, al-

s is heavily
es for later
nce in our
ially written,
eriod of use,
m so that they

the author, with
ur anywhere within
telling where it
the time and date,
and the last in-
has taken place
material should
to sort it on
t various sorted

ent of the dialog,
sible is lost.
ently open to an-
ceed (for a finite
in the program we
error occurs; if

that error indicates a file current in use, we behave as indicated.

It may happen, too, that for one reason or another the file has been destroyed. Here we go to particular pains to let this be known immediately so that we lose as little information as we possibly can. If a response file does not exist when a student tries to write on it, we first try to recreate it. If this fails (usually because we are not in the same account in which the file is to exist), we send a message to the console instructing the computer operator to run a program which will recreate the file, a program which we supply. If this program itself bombs, it asks the computer operator to call someone connected with the project, so that we can take whatever action is possible. Thus we take more than usual precaution to be certain that we lose as little data as possible, since this data is critical for rewriting the programs.

Other kinds of file activities also occur, and are treated similarly. Restarting a student within a program he did not complete is based on a file that stores for each student his sign on number, a core address, the overlay segment currently in use, and the value of all the counters, the things which determine looping within the program. Thus we are able to start a student in the same situation which he left, provided all continuing information is stored in counters. On the Sigma 7 we handle this file as a keyed file, with the key having a part which identifies the program and a part which identifies the student. It is, incidentally, necessary to query a student as to whether he was the one who put in the particular ID before, because experience indicates that with large classes such IDs as "BILL" will be common!

One record keeping activity is connected with both of these, and concerns the presence of errors in the system, both within the programs themselves and in file operations. As indicated when a file error occurs, we make a careful check on the type of error, by inspecting the error code, and we take as many as a half dozen different actions depending on this code. Programming errors are also common when the programs are first released, because they are complex programming and no amount of initial running will reveal all the errors which may be present. As with any complex programming errors may be still present after hundreds of uses and several revisions.

Our philosophy for error messages is that we shield the student almost entirely from such messages. We keep error messages on internal files but we do not tell these to the student. In many cases a student is unaware that any error has occurred because he will simply keep going in the program. If the error is unrecoverable, we dump him out of the program keeping the error information ourself for later use. We believe that nothing turns the student off faster than a computerese error message that is not understandable to him in the context in which he has just been working. Since we work at the assembly language level, we can seize control of all error conditions, by means of our own trap instructions and by using the file error procedures provided by the monitor.

Documentation

One other point that should be kept in mind is that documentation is essential for a full system, and should be considered part of the conversion process. This includes the manuals we now have on hand, including the supplementary sections. To get large numbers of people to work on teaching materials you must describe the facilities at a variety of different levels. Some of our present documentation might go with other implementations, but any implementation is system-dependent and this must be reflected in new and adequately written documentation. In our case we have employed an outside consultant, Chuck Mossman, with special skills in writing to improve documentation, because we believe that such materials are very important.